Procedia Computer Science

Volume 80, 2016, Pages 178–189

ICCS 2016. The International Conference on Computational Science

A Performance Prediction and Analysis Integrated Framework for SpMV on GPUs

Ping Guo and Chung-wei Lee

Department of Computer Science, University of Illinois at Springfield, Springfield, Illinois, USA
Email: {pguo6, clee84}@uis.edu

Abstract

This paper presents unique modeling algorithms of performance prediction for sparse matrix-vector multiplication on GPUs. Based on the algorithms, we develop a framework that is able to predict SpMV kernel performance and to analyze the reported prediction results. We make the following contributions: (1) We provide theoretical basis for the generation of benchmark matrices according to the hardware features of a given specific GPU. (2) Given a sparse matrix, we propose a quantitative method to collect some features representing its matrix settings. (3) We propose four performance modeling algorithms to accurately predict kernel performance for SpMV computing using CSR, ELL, COO, and HYB SpMV kernels. We evaluate the accuracy of our framework with 8 widely-used sparse matrices (totally 32 test cases) on NVIDIA Tesla K80 GPU. In our experiments, the average performance differences between the predicted and measured SpMV kernel execution times for CSR, ELL, COO, and HYB SpMV kernels are 5.1%, 5.3%, 1.7%, and 6.1%, respectively.

Keywords: GPU, performance modeling, sparse matrix-vector multiplication

1 Introduction

Sparse matrix-vector multiplication (SpMV) is an essential operation in solving linear systems and partial differential equations. For many scientific and engineering applications, the matrices are naturally large and sparse with various sparsity characteristics. It is a challenging issue to accurately predict SpMV performance. This paper addresses this challenge by presenting performance modeling algorithms to predict SpMV performance on GPUs.

A sparse matrix is a matrix in which most of the elements are zeros and a few other elements are non-zeros. Bell and Garland [1] proposed and implemented some widely-used formats to store non-zero elements in a sparse matrix, including CSR (Compressed Sparse Row), ELL (ELLPACK), COO (Coordinate), and HYB (Hybrid). They also designed and implemented CUDA-based SpMV computational kernels to perform SpMV operations with each sparse matrix format. From our experiments, we observed that different matrices may have their own most appropriate storage formats to achieve the best performance. This observation motivates

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.308

us to design a framework to provide performance prediction for SpMV computing using multiple SpMV kernels. The prediction results reported by our framework can be used to effectively assist researchers in foreseeing SpMV performance before some specific programs are actually to be run. In addition, our framework can further assist researchers in making choice of appropriate matrix storage formats and test matrices. Specifically, given a target sparse matrix and a specific GPU architecture, the most appropriate matrix storage format among CSR, ELL, COO, and HYB formats can be selected by comparing the predicted kernel performance of four SpMV computational kernels which are proposed and implemented for these four sparse matrix storage formats. In another aspect, given a specific GPU architecture, researchers can select an appropriate small set of test matrices for experiments from a large collection of spare matrices by evaluating the predicted performance of SpMV computing using specific kernels.

We solve the performance approximation problem by offline benchmarking. This approach has wide adaptability for sparse matrices with different sparsity characteristics and it supports any NVIDIA GPU platform. For each GPU platform supported, we only need to generate benchmarks for that platform once. The approach to generate benchmark matrices is easy to follow. In addition, our platform-based benchmarking approach has competive advantage in accuracy of performance prediction compared with traditional analytical modeling approach without using benchmarks.

Our performance modeling approach consists of 5 steps in 2 stages, *i.e.*, benchmark generation, performance measurement, relationship establishment, matrix analysis, and performance prediction. In the offline stage, a series of unique benchmark matrices are generated and an SpMV computation is performed by generating a random vector for each benchmark matrix. Some features representing the matrix settings and performance measured for SpMV computations are collected for establishing some linear relationships for performance prediction. In the online stage, the features of a given target sparse matrix are collected as inputs to instantiate our parameterized relationships for performance estimation.

We make the following contributions: (1) We provide theoretical basis for the generation of benchmark matrices according to the hardware features of a given specific GPU. (2) Given a sparse matrix, we propose a quantitative method to collect some features representing its matrix settings. (3) We propose four performance modeling algorithms to accurately predict kernel performance for SpMV computing using CSR, ELL, COO, and HYB SpMV kernels.

2 Related Work

Bolz *et al.* [2] first implemented SpMV computing on GPUs. There have been some existing modeling approaches focusing on performance prediction for SpMV on GPUs. Dinkins [5] proposed a model for predicting SpMV performance using memory bandwidth requirements and data locality. Guo and Wang [6] presented a cross-architecture performance modeling tool to accurately predict SpMV performance on multiple different target GPUs based on the measured performance on a given reference GPU. Li *et al.* [9] proposed a modeling approach for SpMV performance estimation by analyzing the distribution characteristics of non-zero elements. The modeling approaches focusing on performance optimization includes [3, 7, 8, 12]. Choi *et al.* [3] designed a blocked ELLPACK (BELLPACK) format and proposed a performance model to predict matrix-dependent tuning parameters. Guo *et al.* [7] designed a dynamic-programming algorithm to optimize SpMV performance based on the prediction results reported by modeling tool. Karakasis *et al.* [8] presented a performance model that can accurately select the most suitable blocking sparse matrix storage format and its proper configuration. Xu *et al.* [12] proposed the optimized SpMV based on ELL format and a performance model.

3 Sparse Matrix Formats

CSR, ELL, COO, and HYB (ELL/COO) are four widely-used sparse storage formats, each of which has a corresponding SpMV computational kernel implemented by Bell and Garland [1]. Our research work utilizes their kernels.

- The CSR (Compressed Sparse Row) format is the most popular sparse matrix format. It consists of three arrays: **RowPtr**, **Col**, and **Data**. The integer array **RowPtr** stores row pointers to the offset of each row. The integer array **Col** stores the column indices of non-zero elements. The array **Data** stores the values of non-zero elements.

- The ELL(ELLPACK) format stores a sparse matrix in two arrays: **Data** and **Col**. The array **Data** stores the values of non-zero elements. The integer array **Col** stores the column indices of each non-zero element.

- The COO (Coordinate) format is the most simple and intuitive storage scheme to store a sparse matrix. It is composed of three arrays: **Row**, **Col**, and **Data**. The **Row** array, the **Col** array, and the **Data** array store the row indices, the column indices, and the values of non-zero elements, respectively.

- The HYB (Hybrid ELL/COO) format stores the majority of non-zero elements in ELL format and the remaining non-zero elements in COO format. All non-zero elements at the columns on the left of the threshold value are stored in the ELL format and the rest non-zero elements are stored in the COO format.

A sample sparse matrix M (5 by 4) and its CSR, ELL, COO, and HYB representations are illustrated in Figure 1.

$$M = \begin{bmatrix} 5 & 8 & 0 & 6 \\ 0 & 9 & 4 & 0 \\ 7 & 1 & 6 & 8 \\ 6 & 3 & 0 & 0 \\ 0 & 5 & 0 & 1 \end{bmatrix}$$

Original Sparse Matrix

$$Data = \begin{bmatrix} 5 & 8 & 6 & * \\ 9 & 4 & * & * \\ 7 & 1 & 6 & 8 \\ 6 & 3 & * & * \\ 5 & 1 & * & * \end{bmatrix} \qquad Col = \begin{bmatrix} 0 & 1 & 3 & * \\ 1 & 2 & * & * \\ 0 & 1 & 2 & 3 \\ 0 & 1 & * & * \\ 1 & 3 & * & * \end{bmatrix}$$

ELL Storage Representation

Row = [0 0 0 1 1 2 2 2 2 3 3 4 4]
Col = [0 1 3 1 2 0 1 2 3 0 1 1 3]
Data = [5 8 6 9 4 7 1 6 8 6 3 5 1]

COO Storage Representation

RowPtr = [0 3 5 9 11 13]
Col = [0 1 3 1 2 0 1 2 3 0 1 1 3]
Data = [5 8 6 9 4 7 1 6 8 6 3 5 1]

CSR Storage Representation

Figure 1: Original sparse matrix (left top); COO storage representation (left bottom); ELL storage representation (right top), where "*" represents padding zero; CSR storage representation (right bottom)

4 SpMV Performance Prediction

Our performance modeling approach is composed of two stages. The offline stage consists of 3 steps, *i.e.*, benchmark generation, performance measurement, and relationship establishment. The online stage is composed of 2 steps, *i.e.*, matrix analysis and performance prediction. The modeling workflow of our framework is shown in Figure 2.

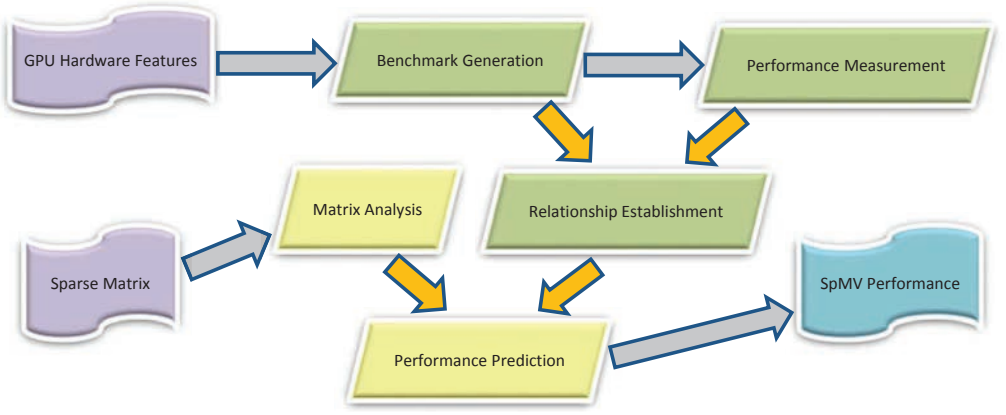


Figure 2: The modeling workflow of our framework

We solve the performance approximation problem by the following way: Given a target sparse matrix M, we first estimate SpMV performance for a sparse matrix B, then use that performance to approximate the performance of matrix M, as shown in Figure 3. The matrix B used to approximate the matrix M has the similar statistical features as that of the matrix M. Specifically, in order to approximate performance of SpMV computing using CSR kernel and ELL kernel, we collect the feature "the number of non-zero rows" from matrix M and set it as that of the matrix B. Therefore, if at least one row with all-zero elements exists in the matrix M, then $R_B < R_M$. Otherwise, $R_B = R_M$. In matrix M, each non-zero row has different number of non-zero elements. Instead, in matrix B, we use a specific value, *i.e.*, the mathematical expectation value, for the performance approximation. Similarly, by the expectation value, the total number of non-zero elements in matrix M is approximated and used for performance estimation of SpMV computing using COO kernel.

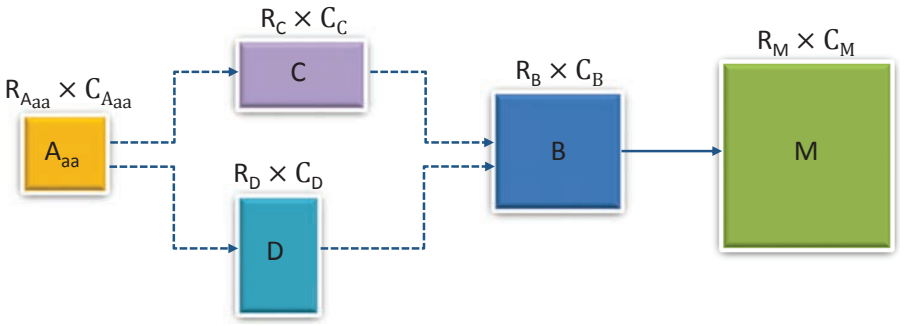


Figure 3: The model of performance approximation

For estimating the performance of SpMV using CSR kernel, the performance of three sparse matrices C, D, and A_{aa} are required for estimating the performance of the sparse matrix B. For two matrices C and D, their performance will be estimated and approximated using some relationships established by the performance of a series of generated benchmark matrices, denoted by A. Among the benchmark matrices in the series of A, the sparse matrix A_{aa} is the benchmark matrix with the minimum dimension generated and used in our models. For matrices B, C, D, and A_{aa}, we let each matrix have a specific number of non-zero elements in each row, denoted by X. In addition, we let $R_D = R_B$, $R_C = R_{A_{aa}}$, $X_D = X_{A_{aa}}$, $X_C = X_B$, $C_B > X_B$, $C_C > X_C$, $C_D > X_D$, and $C_{A_{aa}} > X_{A_{aa}}$. By our modeling approach, the matrix C is not required for the performance approximation of SpMV using ELL kernel, and neither the matrix C nor D is required for the performance estimation of SpMV using COO kernel.

The generation of benchmark matrices in the series of A will be introduced in Section 4.1. For each benchmark matrix, we generate a corresponding random vector to perform SpMV computation. The measured SpMV performance will be used to establish some linear relationships for performance prediction. The measurement of performance and the establishment of relationships will be introduced in Section 4.2 and Section 4.3, respectively. Given a target sparse matrix M, we collect some features and use them to instantiate our parameterized relationships for performance estimation. The collection of features and the estimation of performance will be introduced in Section 4.4 and Section 4.5, respectively.

4.1 Benchmark Generation

Given a benchmark matrix in the series of A, we let each row of the matrix have the same number of non-zero elements by controlling the sparse density. The number of non-zero elements per row, denoted by X, is derived according to the maximum non-zero elements that can be stored in the GPU global memory in the specific sparse matrix format [1]. If we use G_M to represent the size (bytes) of GPU global memory and use R to represent the number of rows of each benchmark matrix in the series of A, its range of value can be represented by: $X \in [1, Max)$. Here, the non-zero elements are in single-precision (float). For double precision, just replacing *float* with *double*.

$$Max = \begin{cases} [G_M - sizeof(int) \times (R+1)]/(sizeof(float) + sizeof(int)) \times R, & \text{for CSR} \\ G_M/[(sizeof(float) + sizeof(int)) \times R], & \text{for ELL} \\ G_M/[(sizeof(float) + 2 \times sizeof(int)) \times R], & \text{for COO} \end{cases} \quad (1)$$

In order to approximate SpMV performance using CSR and ELL kernels, two groups of the benchmark matrices are required to be generated in the series of A. Each group includes the same matrix A_{aa}. The matrices A_{ab}, A_{ac}, ..., A_{al} are in group A_1 and the matrices A_{ba}, A_{ca}, ..., A_{ja} are in group A_2. These matrices are generated from the matrix A_{aa} by varying the values of X and R. Specifically, each matrix in group A_1 takes the same value of R but a different value of X and each matrix in group A_2 takes the same value of X but a different value of R. By our modeling approach, the values of X are chosen in an exponential scale, ranging from 2^2 to 2^{13}. To approximate performance using COO kernel, only group A_1 is required and X takes the values 10, 20, 30, ..., 100 for each benchmark matrix in group A_1, respectively.

The benchmark matrices in the series of A are used for performance approximation. In order to establish linear relationships among the performance of these benchmark matrices, some specific values are assigned to R. The size of R is determined by $R = I \times S$, where $I = 1, 2, ..., 10$ and S represents the maximum number of rows that can be handled by a GPU

with a full load of thread blocks within one iteration [3]. For matrices in the series of A_1, each one takes the value $I = 1$. For benchmark matrices in the series of A_2, each one takes the value of I in order from 1 to 10. When $I = 1$, $R = R_{A_{aa}}$. The size of S is determined by the physical limitations of a GPU and the specific SpMV kernels.

$$S = \begin{cases} N_{SM} \times Warps/Multiprocessor, & \text{for CSR} \\ N_{SM} \times Threads/Multiprocessor, & \text{for ELL \& COO} \end{cases} \quad (2)$$

where,

- N_{SM} represents the number of streaming multiprocessors (SMXs).
- $Warps/Multiprocessor$ represents the number of warps per SMX.
- $Threads/Multiprocessor$ represents the number of threads per SMX.

4.2 Performance Measurement

For each benchmark matrix, we need to perform SpMV computing and measure its performance. Given a benchmark matrix $A'_{m\times n}$, to perform matrix-vector product correctly, the dimension of the random vector V generated for SpMV computing must be n by 1. To ensure that the recorded performance is reliable and stable, we remove the effect of long initialization delay by measuring performance for multiple times and taking an average value. Specifically, firstly, we measure the performance of SpMV computing for continuous α times and β times, respectively, where $\beta > \alpha > 0$ and the performance for the measurement of α times and β times are denoted by T_α and T_β, respectively. Finally, the recorded performance, represented by T, is measured by $T = (T_\beta - T_\alpha)/(\beta - \alpha)$.

4.3 Relationship Establishment

Given a sparse matrix, its performance of SpMV computing using specific kernels can be estimated according to the following relationships. These relationships are established by the statistical features collected from the generated benchmark matrices in the series of A and the measured performance of SpMV computing with these benchmark matrices. Specifically, to predict performance of SpMV computing using CSR kernel and ELL kernel, for each benchmark matrix in group A_1, we collect its feature "the number of non-zero elements per row (X)" and establish the Relationship-1, while for each matrix in group A_2, we collect its feature "the number of rows (R)" and establish the Relationship-2. To predict performance of SpMV computing using COO kernel, for each benchmark matrix in group A_1, we collect its feature "the number of non-zero elements (N_{NZ})" and establish the Relationship-3. Since the HYB is the combination of ELL and COO, to predict performance of SpMV computing using HYB kernel, we will reuse the relationships established for ELL and COO.

- Relationship-1 ($T = L_1(X)$): A linear relationship (L_1) between the number of non-zero elements per row (X) and the performance of SpMV computing (T).
- Relationship-2 ($T = L_2(R)$): A linear relationship (L_2) between the number of rows (R) and the performance of SpMV computing (T).
- Relationship-3 ($T = L_3(N_{NZ})$): A linear relationship (L_3) between the number of non-zero elements (N_{NZ}) and the performance of SpMV computing (T).

4.4 Matrix Analysis

Given a sparse matrix A with M rows and N columns, we let G_i store the number of rows, each of which has i non-zero elements, where $G_i \geq 0$, $i = 0, 1, 2, ..., N$. G_i is a member of a set G, where $G = \{G_0, G_1, ..., G_N\}$. Suppose the matrix A is as shown in Figure 4, then we have $G_2 = G_5 = 2, G_3 = 4, G_4 = G_7 = 1$, and $G_0 = G_1 = G_6 = G_8 = G_9 = G_{10} = 0$.

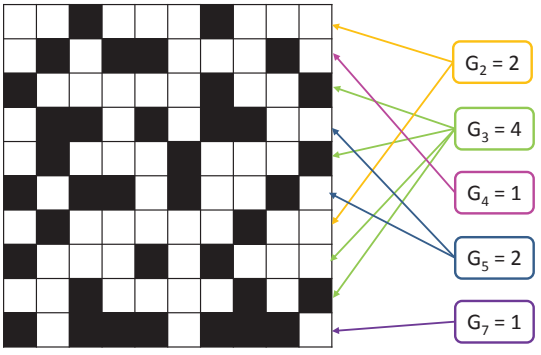


Figure 4: The sparse matrix A (10×10), where each black entry represents a non-zero element

A random variable $X' \in [0, 1, 2, ..., N]$ is defined to represent the number of non-zero elements in one row of matrix A. If a row has i non-zero elements, the variable X' takes the value i, where $i = 0, 1, 2, ..., N$. The probability of such an event $\{X' = \mathrm{i}\}$ happening is represented by P_i, where $P_i \geq 0$, $i = 0, 1, 2, ..., N$.

The relationship between G_i and P_i is expressed by the equation $G_i = M \times P_i$. The set G and the probability P are mathematically characterized by the following equations (3) and (4).

$$M = G_0 + G_1 + ... + G_N = \sum_{i=0}^{N} G_i \tag{3}$$

$$\sum_{i=0}^{N} P_i = P_0 + P_1 + ... + P_N = 1 \tag{4}$$

The mathematical expectation of X', denoted by $E(X')$, is expressed by

$$E(X') = \sum_{i=0}^{N} (i \times P_i) = \frac{1}{M} \times \sum_{i=0}^{N} (i \times G_i) \tag{5}$$

The total number of non-zero elements, represented by N_{NZ}, is expressed by

$$N_{NZ} = E(X') \times M = \sum_{i=0}^{N} (i \times G_i) \tag{6}$$

The number of rows, in each of which the number of non-zero elements X' is greater than a given number K, represented by $M(X' > K)$, is expressed by

$$M(X' > K) = G_{K+1} + G_{K+2} + ... + G_N = \sum_{i=K+1}^{N} G_i \tag{7}$$

For all the rows with $X' > K$, the accumulated sum of the number of non-zero elements in one row X' exceeds K, represented by $N_{NZ}(X' - K)$, is expressed by

$$N_{NZ}(X' - K) = N_{NZ} - K \times M(X' > K) = \sum_{i=0}^{N}(i \times G_i) - K \times \sum_{i=K+1}^{N} G_i \quad (8)$$

4.5 Performance Prediction

4.5.1 Prediction for CSR

Given a sparse matrix M, as shown in Figure 3, we estimate its SpMV performance using CSR kernel by following the steps as follows:

- Estimate the execution time T_1 by $T_1 = L_1(E_0)$ from Relationship-1, where
  - $\diamond$ E_0: The calculated expectation value of X' in matrix M by the equation (5)
  - $\diamond$ T_1: The estimated execution time of SpMV computing with matrix C
- Estimate the execution time T_2 by $T_2 = L_2(R_0)$ from Relationship-2, where
  - $\diamond$ R_0: The number of non-zero rows in matrix M
  - $\diamond$ T_2: The estimated execution time of SpMV computing with matrix D
- Estimate the execution time T_0 by $T_0 = (T_1/T_3) \times T_2$, where
  - $\diamond$ T_0: The estimated execution time of SpMV computing with matrix B
  - $\diamond$ T_3: The measured execution time of SpMV computing with benchmark matrix A_{aa}
- Approximate the execution time T by $T = T_0$, where T represents the estimated execution time of SpMV computing with our target sparse matrix M.

4.5.2 Prediction for ELL

Given a sparse matrix M, as shown in Figure 3, we estimate its SpMV performance using ELL kernel by following the steps as follows:

- Estimate the execution time T_1 by $T_1 = L_2(R_0)$ from Relationship-2, where
  - $\diamond$ R_0: The number of non-zero rows in matrix M
  - $\diamond$ T_1: The estimated execution time of SpMV computing with matrix D
- Obtain the linear equation $L_1' : Y = L'(a)X + L'(b)$, where
  - $\diamond$ $L'(a)$: The coefficient of the linear equation L_1', which is equal to $R_0/R_{A_{aa}}$ times of the coefficient of the linear equation L_1 in Relationship-1

- ◇ $L'(b)$: The intercept of the linear equation L'_1, which is equal to $T_1 - L'(a) \times X_{A_{aa}}$
- ◇ L'_1: The new equation of Relationship-1, which is established for matrices in group A_1 but changing the number of rows from $R_{A_{aa}}$ to R_0

- Estimate the execution time T_0 by $T_0 = L'(a) \times E_0 + L'(b)$, where E_0 represents the calculated expectation value of X' in matrix M by the equation (5).
- Approximate the execution time T by $T = T_0$, where T represents the estimated execution time of SpMV computing with our target sparse matrix M.

4.5.3 Prediction for COO

Given a sparse matrix M, as shown in Figure 3, we estimate its SpMV performance using COO kernel by following the steps as follows:

- Estimate the total number of non-zero elements (N_{NZ}) in matrix M by the equation (6).
- Estimate the execution time T_0 using Relationship-3 by $T_0 = L_3(N_{NZ})$, where T_0 represents the estimated execution time of SpMV computing with matrix B.
- Approximate the execution time T by $T = T_0$, where T represents the estimated execution time of SpMV computing with our target sparse matrix M.

4.5.4 Prediction for HYB

Given a sparse matrix M, we estimate its SpMV performance using HYB kernel by following the steps as follows:

- Divide the matrix M into two submatrices by HYB threshold K [1]: ELL and COO.
- Estimate the execution time of T_1 by $T_1 = L'(a) \times K + L'(b)$ using the method mentioned in the section 4.5.2, where T_1 represents the estimated execution time of SpMV computing with ELL submatrix.
- Estimate the execution time T_2 by $T_2 = L_3(N'_{NZ})$ from Relationship-3, where
  - ◇ N'_{NZ}: The number of non-zero elements of the COO submatrix, which is computed by the equation (8).
  - ◇ T_2: The estimated execution time of SpMV computing with COO submatrix
- Approximate the execution time T_0 by $T_0 = T_1 + T_2$, where T_0 represents the estimated execution time of SpMV computing with matrix B.
- Approximate the execution time T by $T = T_0$, where T represents the estimated execution time of SpMV computing with our target sparse matrix M.

5 Experimental Evaluation

5.1 Experimental Settings

Our experiments are conducted at comet cluster supported by $XSEDE$ [10]. Comet is equipped with 1944 standard compute nodes. Each Comet standard node will have two Intel Xeon E5-2680v3 processors, 128 GB of DDR4 memory, and two 160-GB flash drives. Thirty-six of the standard nodes will each be augmented with four NVIDIA Tesla K80 GPUs, whose compute capability is 3.7. As for software, our experiments uses CUDA 7.0. To evaluate the accuracy of our performance modeling, we compare performance reported by our performance models with performance measured from NVIDIA's implementation. In our experiments, all the kernel execution times measured from NVIDIA's implementation are averaged after running 300 times. The sparse matrices used in our experiments are chosen from [4, 11], whose features are shown in Table 1. These matrices are from a wide variety of real applications.

Table 1: Sparse matrices used in our experimental evaluation.

Matrix	Dimensions	NZs	Matrix	Dimensions	NZs
Dense	2K*2K	4.0M	FEM/Harbor	47K*47K	2.37M
Protein	36K*36K	4.3M	QCD	49K*49K	1.90M
FEM/Spheres	83K*83K	6.0M	FEM/Ship	141K*141K	3.98M
FEM/Cantilever	62K*62K	4.0M	Epidemiology	526K*526K	2.1M

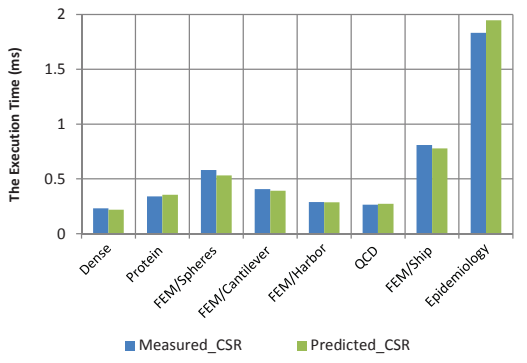
5.2 Results and Analysis

We evaluate the accuracy of the prediction results reported by our framework. The experiments are conducted on 8 matrices for SpMV computing using CSR, ELL, COO, and HYB kernels. Hence, there are 32 test cases in total. The measured SpMV kernel execution times on the NVIDIA Tesla K80, represented by blue bars, and the predicted execution times reported by our framework, represented by green bars, are shown in Figure 5. Specifically, the performance comparisons between the measured and predicted execution times for SpMV computing using CSR, ELL, COO, and HYB kernels are shown in Figures 5(a), 5(b) 5(c), and 5(d), respectively. The average performance differences between the predicted and measured execution times for CSR, ELL, COO, and HYB SpMV kernels are 5.1%, 5.3%, 1.7%, and 6.1%, respectively.

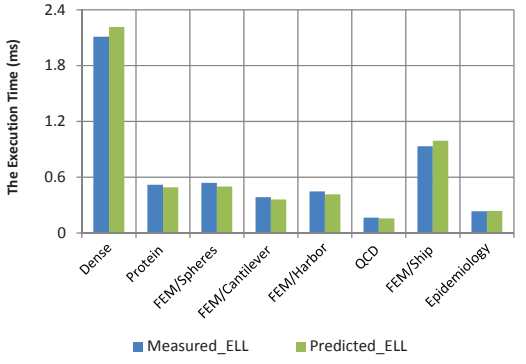
The prediction results reported by our framework can be used to effectively assist researchers in foreseeing SpMV performance before some specific programs are actually to be run. In addition, our framework can further assist researchers in making choice of appropriate matrix storage formats and test matrices. Specifically, given a target sparse matrix and a specific GPU architecture, *e.g.*, NVIDIA Tesla K80 GPU, the most appropriate matrix storage format among CSR, ELL, COO, and HYB formats can be selected by comparing the predicted kernel performance. In another aspect, given a specific GPU architecture, researchers can select an appropriate small set of test matrices for experiments from a large collection of spare matrices by evaluating the predicted performance of SpMV computing using specific kernels.

6 Conclusions

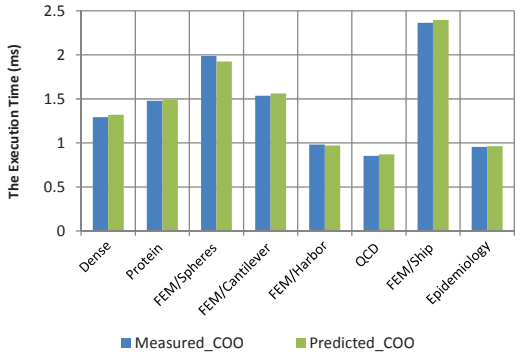
In this paper, we develop a framework that is able to predict SpMV kernel performance and to analyze the reported prediction results. We solve the performance approximation problem by



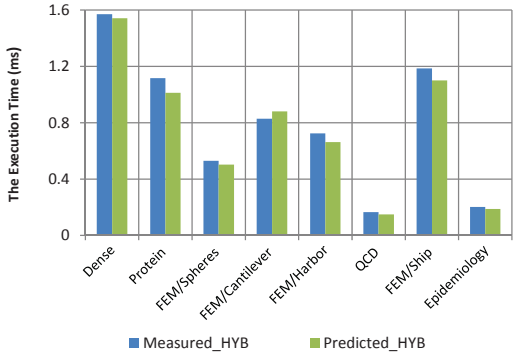
(a) Evaluation on CSR kernel



(b) Evaluation on ELL kernel



(c) Evaluation on COO kernel



(d) Evaluation on HYB kernel

Figure 5: Accuracy evaluation on NVIDIA Tesla K80

offline benchmarking. This approach has wide adaptability for sparse matrices with different sparsity characteristics and it supports any NVIDIA GPU platform. We make the following contributions: (1) We provide theoretical basis for the generation of benchmark matrices according to the hardware features of a given specific GPU. (2) Given a sparse matrix, we propose a quantitative method to collect some features representing its matrix settings. (3) We propose four performance modeling algorithms to accurately predict kernel performance for SpMV computing using CSR, ELL, COO, and HYB SpMV kernels. The prediction results can be used to effectively assist researchers in foreseeing SpMV performance before some specific programs are actually to be run. In addition, our framework can further assist researchers in making choice of appropriate matrix storage formats and test matrices. We can envision this approach being deployed as a part of the high performance computing (HPC) service from cloud computing providers such as the Amazon Elastic Compute Cloud (EC2).

7 Acknowledgments

This research was made possible by the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

References

[1] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 18:1–18:11. ACM, 2009.

[2] J. Bolz, I. Farmer, E. Grinspun, and P. Schroder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph.*, 22(3):917–924, 2003.

[3] J. W. Choi, A. Singh, and R. W. Vuduc. Model-driven autotuning of sparse matrix-vector multiply on GPUs. In *PPoPP '10: Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '10, pages 115–126. ACM, 2010.

[4] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, 2011.

[5] Stephanie Dinkins. A model for predicting the performance of sparse matrix vector multiply (SpMV) using memory bandwidth requirements and data locality. Master's thesis, Colorado State University, 2012.

[6] Ping Guo and Liqiang Wang. Accurate cross-architecture performance modeling for sparse matrix-vector multiplication (SpMV) on GPUs. *Concurrency and Computation: Practice and Experience*, 2014.

[7] Ping Guo, Liqiang Wang, and Po Chen. A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1112–1123, 2014.

[8] Vasileios Karakasis, Georgios Goumas, and Nectarios Koziris. Perfomance models for blocked sparse matrix-vector multiplication kernels. In *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, pages 356–364. IEEE Computer Society, 2009.

[9] Kenli Li, Wangdong Yang, and Keqin Li. Performance analysis and optimization for spmv on gpu using probabilistic modeling. *Parallel and Distributed Systems, IEEE Transactions on*, 26(1):196–205, Jan 2015.

[10] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D. Peterson, Ralph Roskies, J. Ray Scott, and Nancy Wilkens-Diehr. Xsede: Accelerating scientific discovery. *Computing in Science and Engineering*, 16(5):62–74, 2014.

[11] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proc. 2007 ACM/IEEE Conference on Supercomputing*, SC '07, pages 38:1–38:12. ACM, 2007.

[12] Shiming Xu, Wei Xue, and HaiXiang Lin. Performance modeling and optimization of sparse matrix-vector multiplication on NVIDIA CUDA platform. *The Journal of Supercomputing*, 63:710–721, 2013.